12

Variations on Backpropagation

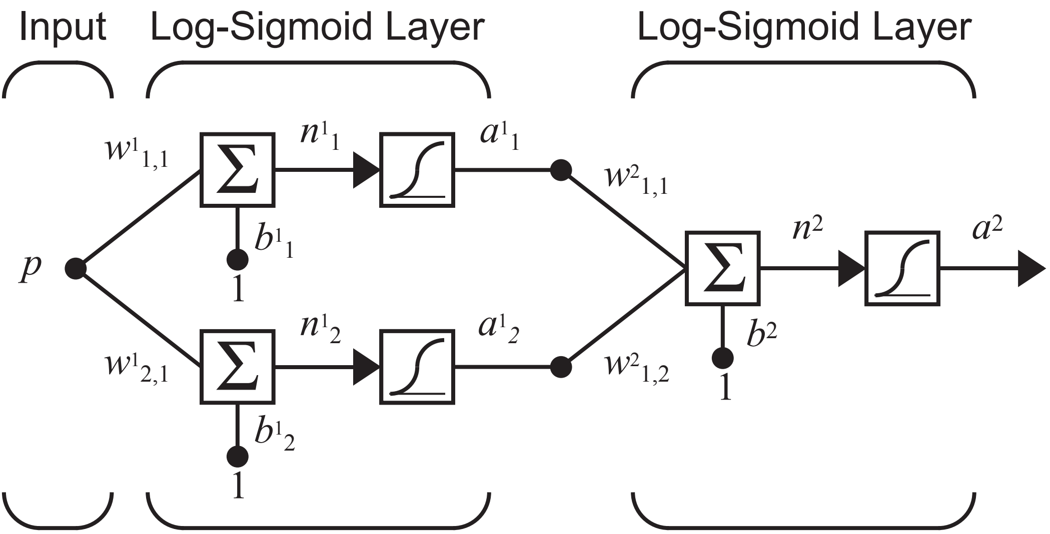
Variations

- Heuristic Modifications
  - Momentum
  - Variable Learning Rate

- Standard Numerical Optimization
  - Conjugate Gradient
  - Newton’s Method (Levenberg-Marquardt)

Performance Surface Example

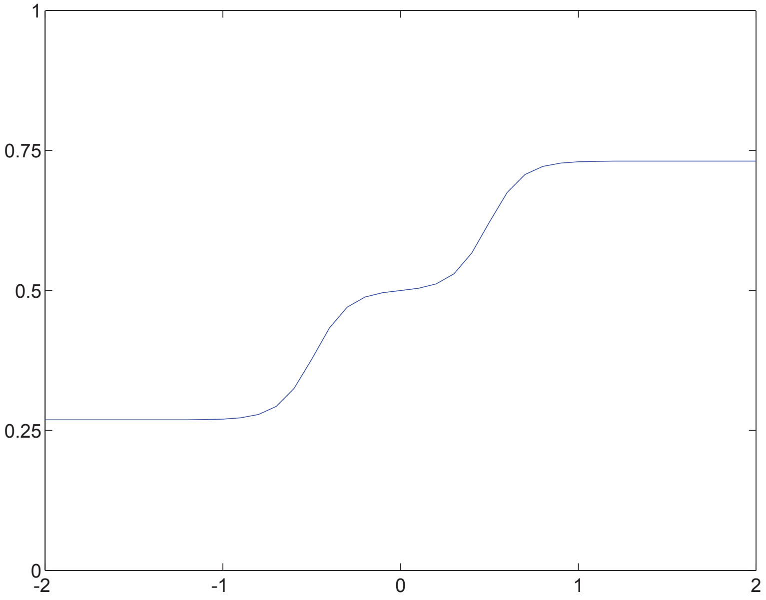
Network Architecture



$\mathbf{a}^1 = \mathbf{logsig}(\mathbf{W}^1 p + \mathbf{b}^1)$

$a^2 = logsig(\mathbf{W}^2 \mathbf{a}^1 + b^2)$

Nominal Function

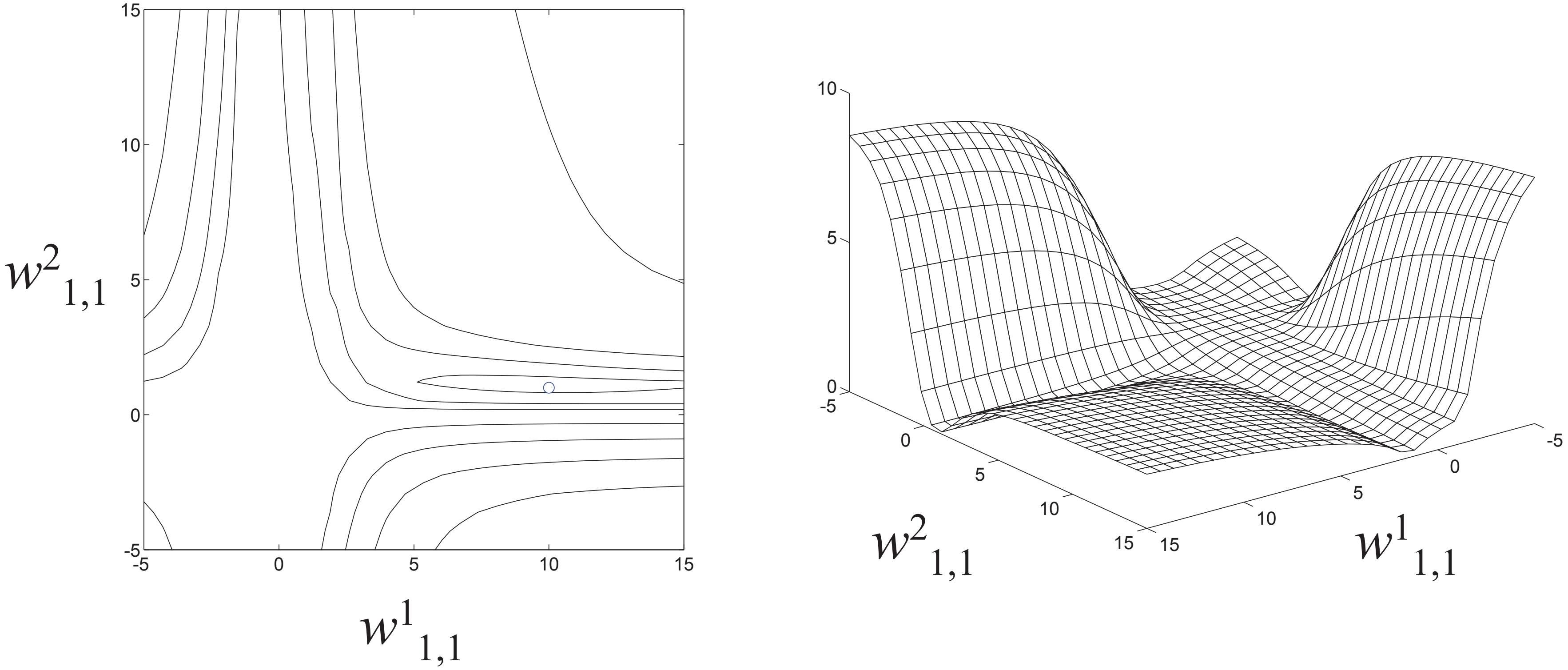


Parameter Values

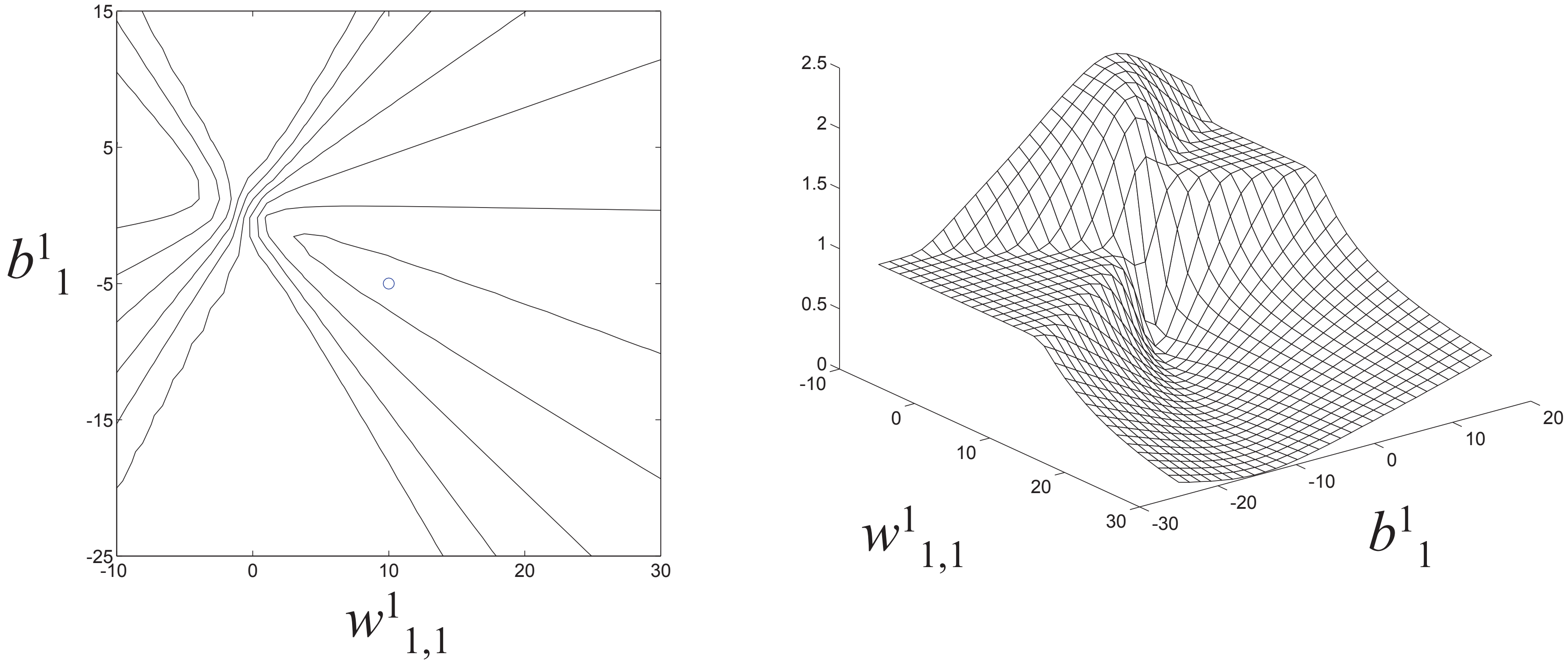
$$w^1_{1,1} = 10 \qquad w^1_{2,1} = 10 \qquad b^1_1 = -5 \qquad b^1_2 = 5$$

$$w^2_{1,1} = 1 \qquad w^2_{1,2} = 1 \qquad b^2 = -1$$

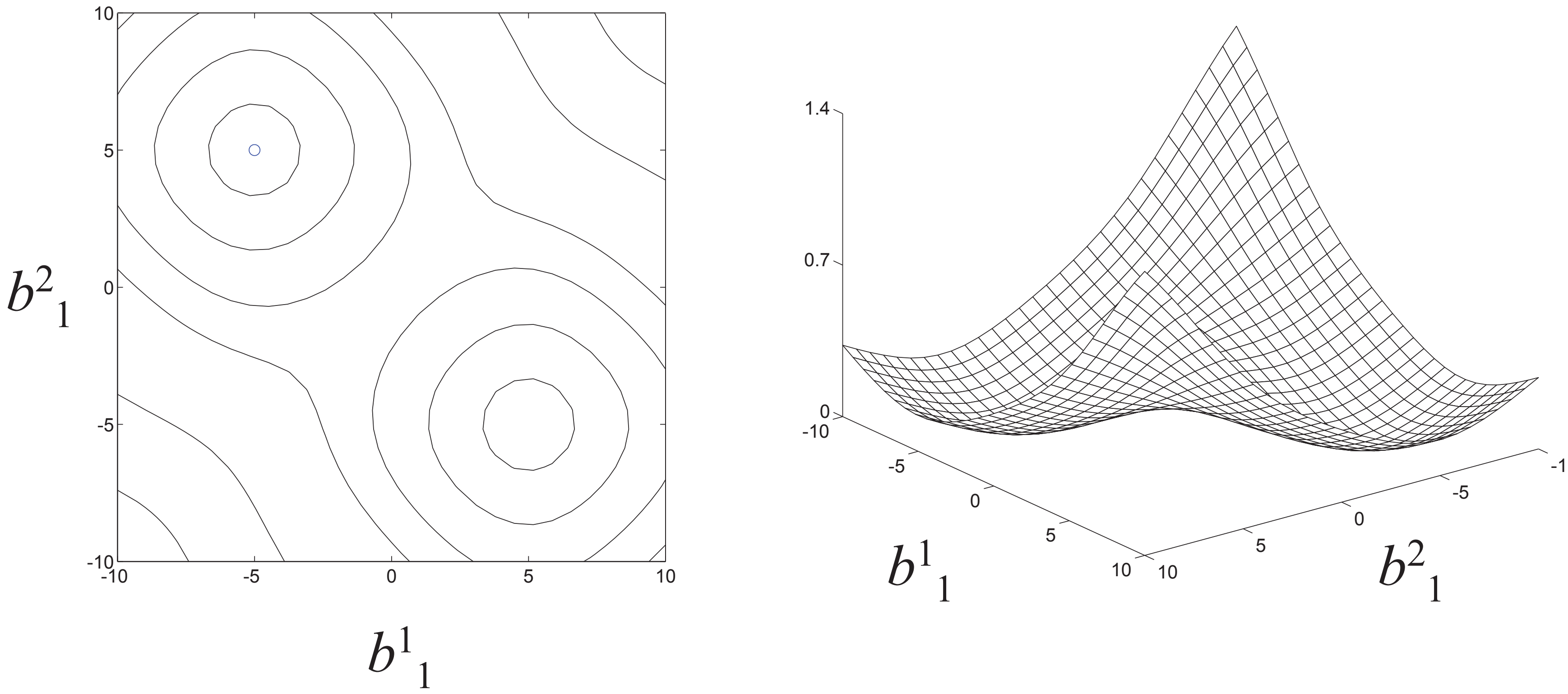
Squared Error vs. $w^1_{1,1}$ and $w^2_{1,1}$



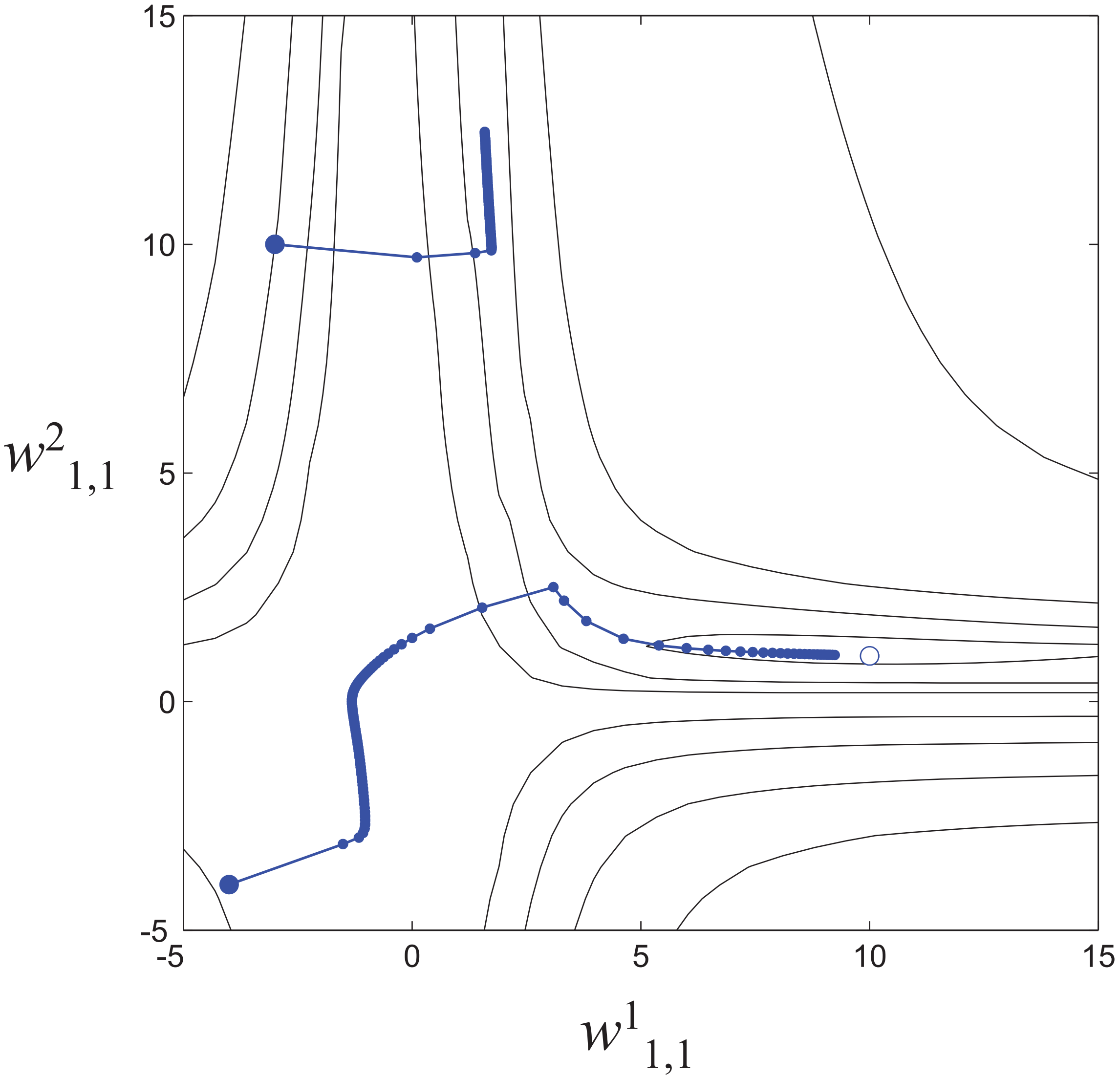
Squared Error vs. $w^1_{1,1}$ and b^1_1



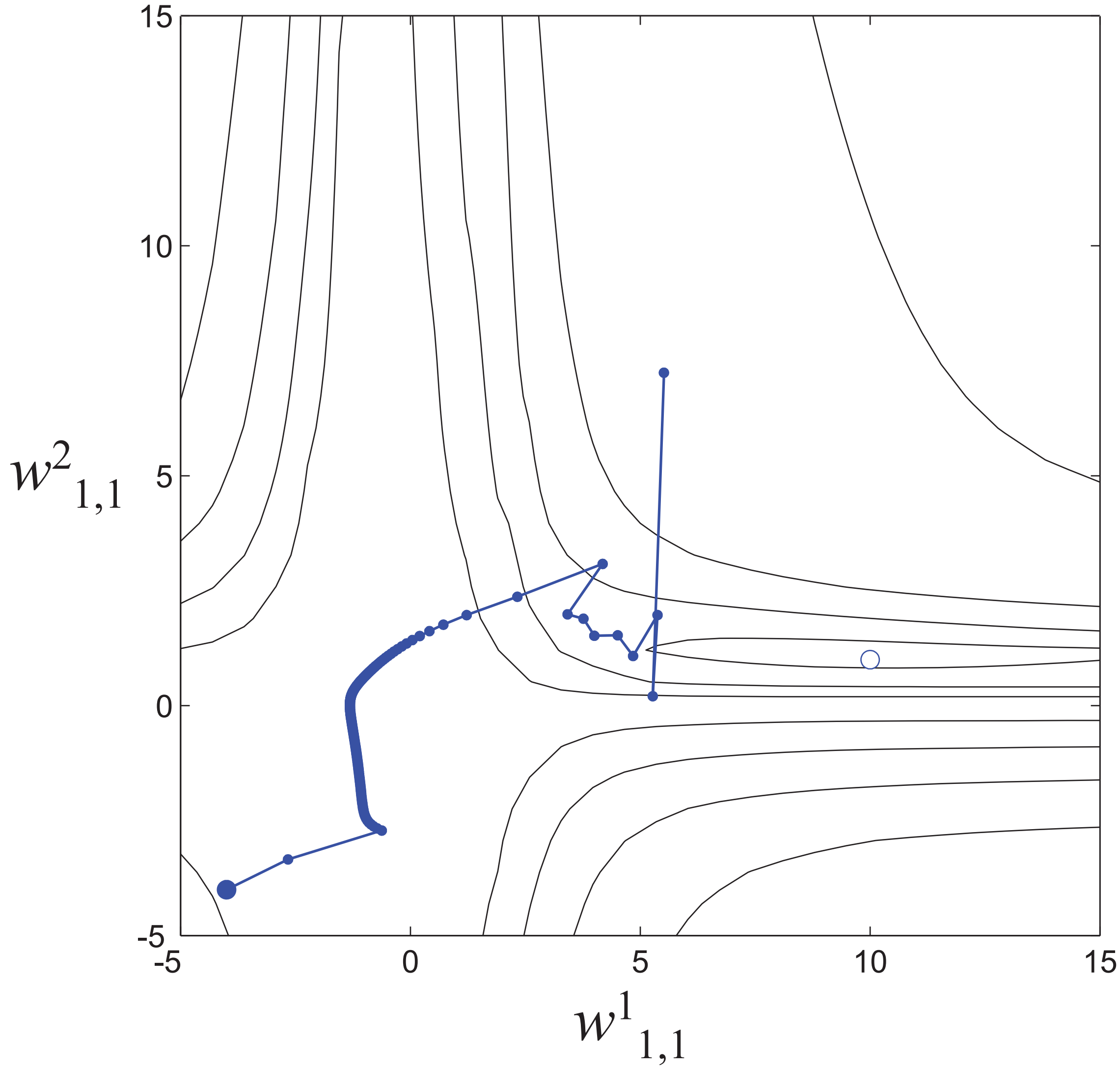
Squared Error vs. b^1_1 and b^1_2



Convergence Example



Learning Rate Too Large



Momentum

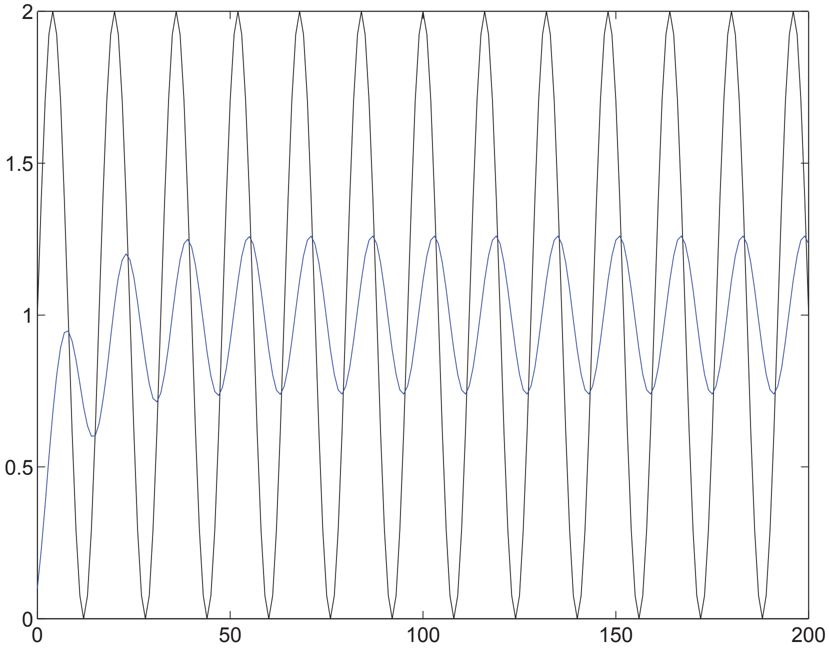
Filter

$$y(k) = \gamma y(k-1) + (1-\gamma)w(k) \qquad 0 \leq \gamma < 1$$

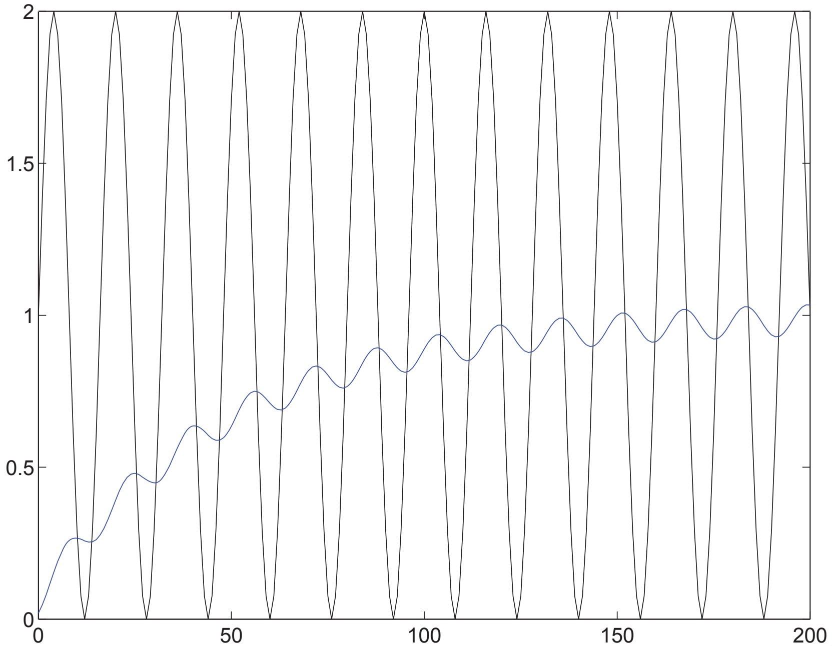
Example

$$w(k) = 1 + \sin\left(\frac{2\pi k}{16}\right)$$

$\gamma = 0.9$



$\gamma = 0.98$



Momentum Backpropagation

Steepest Descent Backpropagation (SDBP)

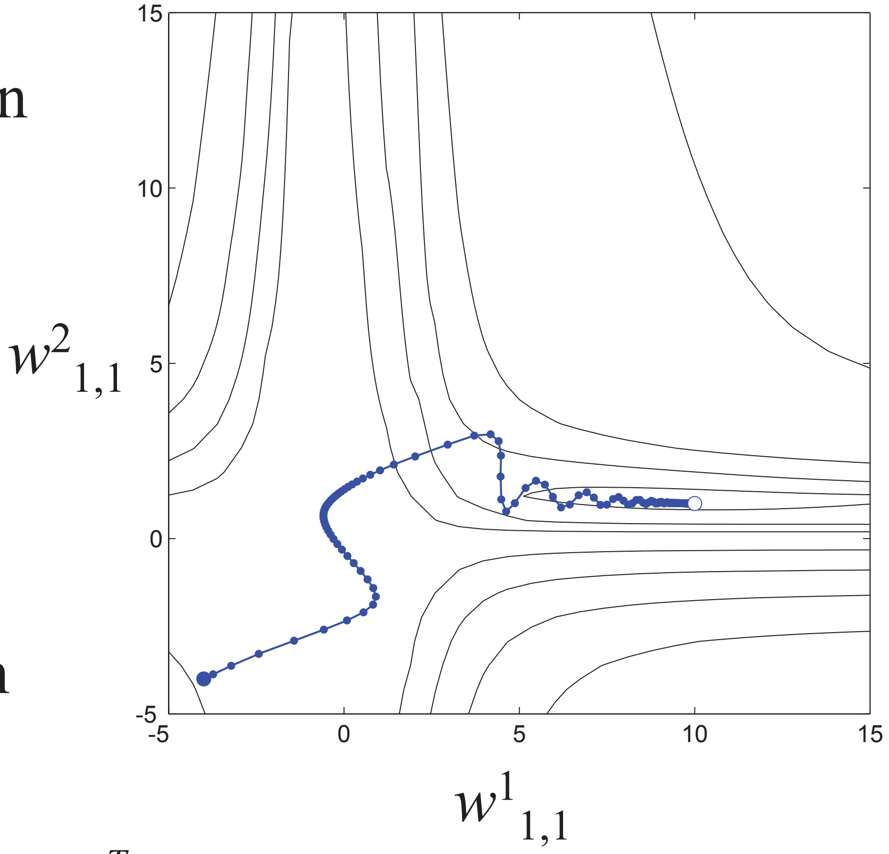
$$\Delta \mathbf{W}^m(k) = -\alpha \mathbf{s}^m (\mathbf{a}^{m-1})^T$$

$$\Delta \mathbf{b}^m(k) = -\alpha \mathbf{s}^m$$

Momentum Backpropagation (MOBP)

$$\Delta \mathbf{W}^m(k) = \gamma \Delta \mathbf{W}^m(k-1) - (1-\gamma)\alpha \mathbf{s}^m (\mathbf{a}^{m-1})^T$$

$$\Delta \mathbf{b}^m(k) = \gamma \Delta \mathbf{b}^m(k-1) - (1-\gamma)\alpha \mathbf{s}^m$$

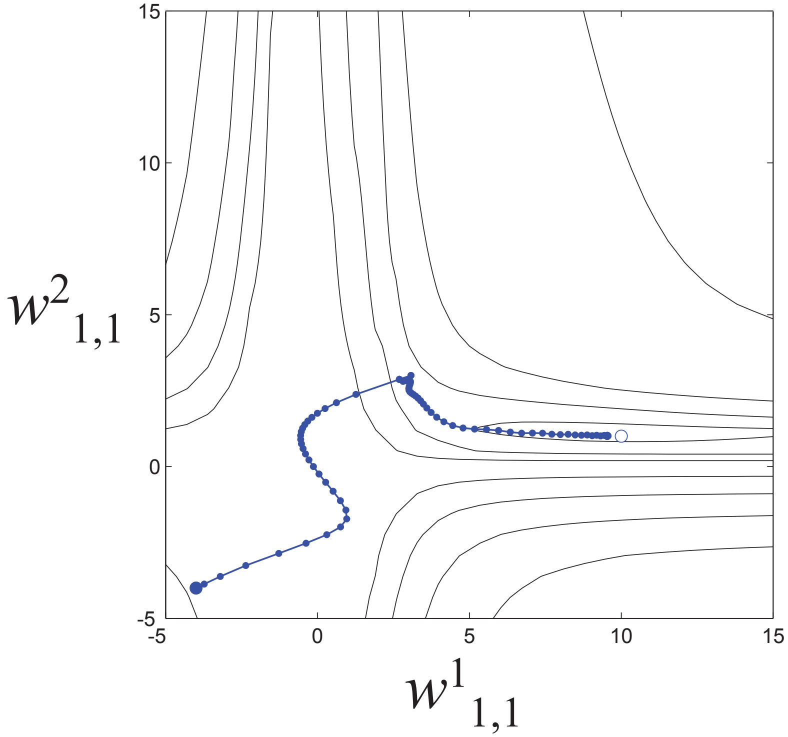


$$\gamma = 0.8$$

Variable Learning Rate (VLBP)

- If the squared error (over the entire training set) increases by more than some set percentage ζ after a weight update, then the weight update is discarded, the learning rate is multiplied by some factor $(1>\rho>0)$, and the momentum coefficient γ is set to zero.
- If the squared error decreases after a weight update, then the weight update is accepted and the learning rate is multiplied by some factor $\eta>1$. If γ has been previously set to zero, it is reset to its original value.
- If the squared error increases by less than ζ, then the weight update is accepted, but the learning rate and the momentum coefficient are unchanged.

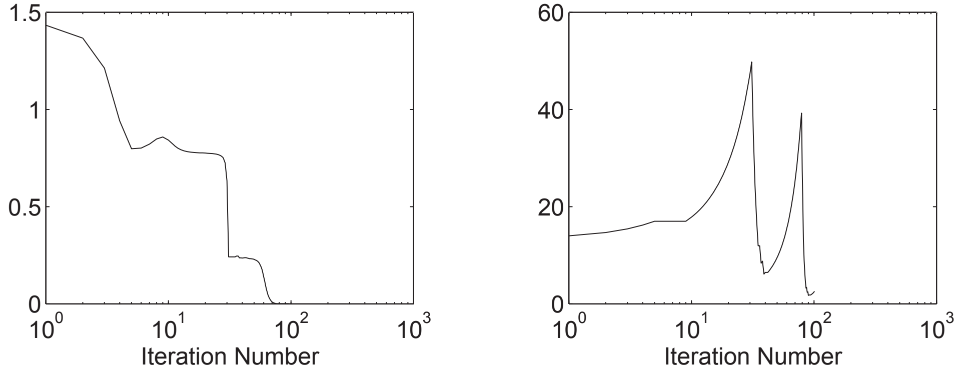
Example



$\eta = 1.05$

$\rho = 0.7$

$\zeta = 4\%$



Conjugate Gradient

1. The first search direction is steepest descent.

$$\mathbf{p}_0 = -\mathbf{g}_0 \qquad \mathbf{g}_k \equiv \nabla F(\mathbf{x})\Big|_{\mathbf{x} = \mathbf{x}_k}$$

2. Take a step and choose the learning rate to minimize the function along the search direction.

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$$

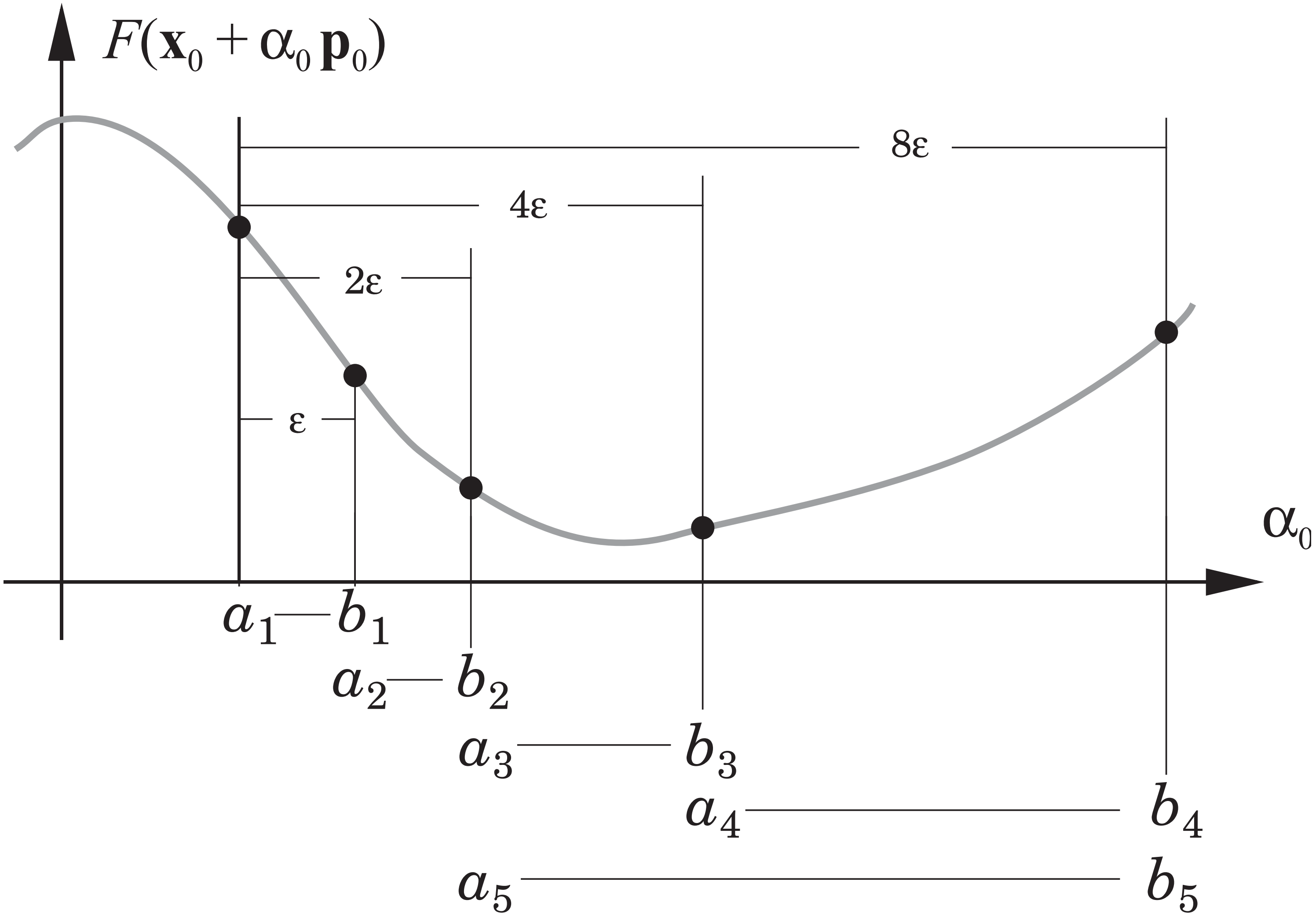
3. Select the next search direction according to:

$$\mathbf{p}_k = -\mathbf{g}_k + \beta_k \mathbf{p}_{k-1}$$

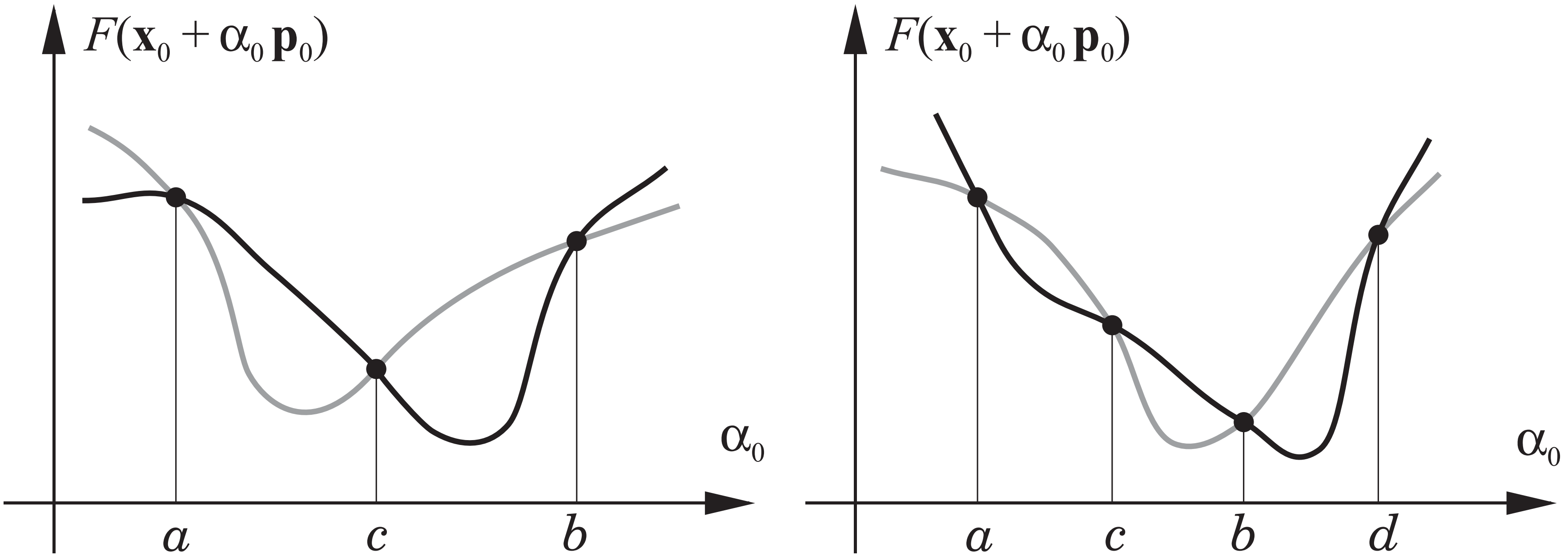
where

$$\beta_k = \frac{\Delta\mathbf{g}_{k-1}^T \mathbf{g}_k}{\Delta\mathbf{g}_{k-1}^T \mathbf{p}_{k-1}} \quad \text{or} \quad \beta_k = \frac{\mathbf{g}_k^T \mathbf{g}_k}{\mathbf{g}_{k-1}^T \mathbf{g}_{k-1}} \quad \text{or} \quad \beta_k = \frac{\Delta\mathbf{g}_{k-1}^T \mathbf{g}_k}{\mathbf{g}_{k-1}^T \mathbf{g}_{k-1}}$$

Interval Location



Interval Reduction



(a) Interval is not reduced.

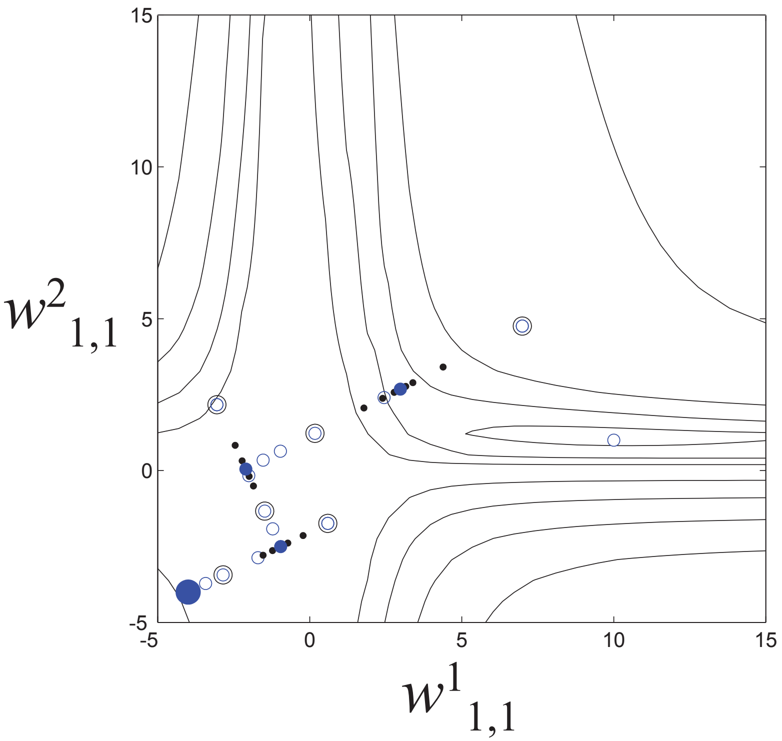
(b) Minimum must occur between *c* and *b*.

Golden Section Search

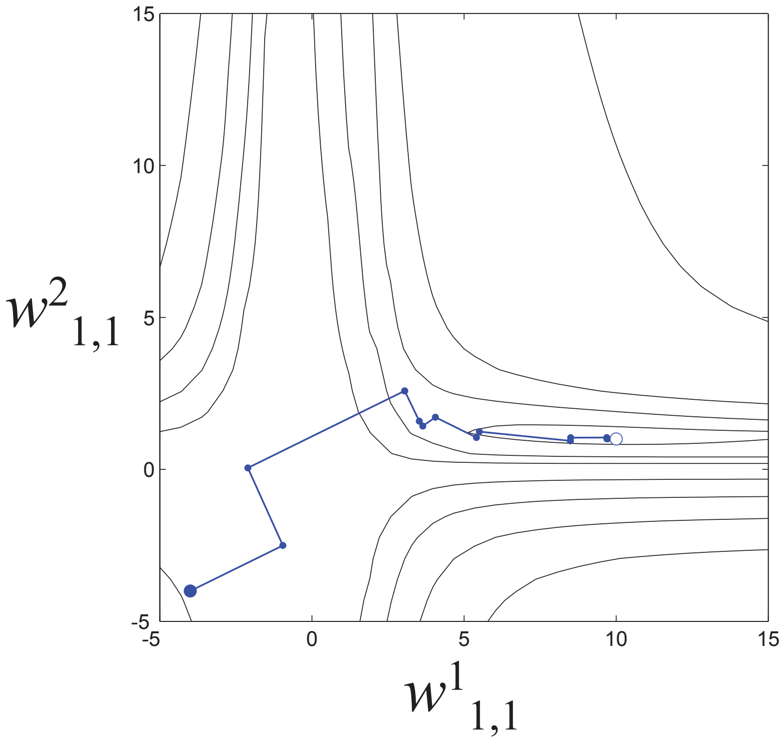
```
$\tau$=0.618
Set   $c_1 = a_1 + (1-\tau)(b_1-a_1)$, $F_c=F(c_1)$
      $d_1 = b_1 - (1-\tau)(b_1-a_1)$, $F_d=F(d_1)$
For $k$=1,2, ... repeat
    If $F_c < F_d$ then
        Set   $a_{k+1} = a_k$ ; $b_{k+1} = d_k$ ; $d_{k+1} = c_k$
              $c_{k+1} = a_{k+1} + (1-\tau)(b_{k+1} - a_{k+1})$
              $F_d = F_c$; $F_c=F(c_{k+1})$
    else
        Set   $a_{k+1} = c_k$ ; $b_{k+1} = b_k$ ; $c_{k+1} = d_k$
              $d_{k+1} = b_{k+1} - (1-\tau)(b_{k+1} - a_{k+1})$
              $F_c = F_d$; $F_d=F(d_{k+1})$
    end
end until $b_{k+1} - a_{k+1} < tol$
```

Conjugate Gradient BP (CGBP)

Intermediate Steps



Complete Trajectory



Newton's Method

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{A}_k^{-1}\mathbf{g}_k$$

$$\mathbf{A}_k \equiv \nabla^2 F(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}_k} \qquad \mathbf{g}_k \equiv \nabla F(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}_k}$$

If the performance index is a sum of squares function:

$$F(\mathbf{x}) = \sum_{i=1}^{N} v_i^2(\mathbf{x}) = \mathbf{v}^T(\mathbf{x})\mathbf{v}(\mathbf{x})$$

then the *j*th element of the gradient is

$$[\nabla F(\mathbf{x})]_j = \frac{\partial F(\mathbf{x})}{\partial x_j} = 2\sum_{i=1}^{N} v_i(\mathbf{x})\frac{\partial v_i(\mathbf{x})}{\partial x_j}$$

Matrix Form

The gradient can be written in matrix form:

$$\nabla F(\mathbf{x}) = 2\mathbf{J}^T(\mathbf{x})\mathbf{v}(\mathbf{x})$$

where **J** is the Jacobian matrix:

$$\mathbf{J}(\mathbf{x}) = \begin{bmatrix} \frac{\partial v_1(\mathbf{x})}{\partial x_1} & \frac{\partial v_1(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial v_1(\mathbf{x})}{\partial x_n} \\ \frac{\partial v_2(\mathbf{x})}{\partial x_1} & \frac{\partial v_2(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial v_2(\mathbf{x})}{\partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial v_N(\mathbf{x})}{\partial x_1} & \frac{\partial v_N(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial v_N(\mathbf{x})}{\partial x_n} \end{bmatrix}$$

Hessian

$$[\nabla^2 F(\mathbf{x})]_{\mathrm{k,\,j}} = \frac{\partial^2 F(\mathbf{x})}{\partial x_k \partial x_j} = 2\sum_{i=1}^{N}\left\{\frac{\partial v_i(\mathbf{x})}{\partial x_k}\frac{\partial v_i(\mathbf{x})}{\partial x_j} + v_i(\mathbf{x})\frac{\partial^2 v_i(\mathbf{x})}{\partial x_k \partial x_j}\right\}$$

$$\nabla^2 \mathrm{F}(\mathbf{x}) = 2\mathbf{J}^T(\mathbf{x})\mathbf{J}(\mathbf{x}) + 2\mathbf{S}(\mathbf{x})$$

$$\mathbf{S}(\mathbf{x}) = \sum_{i=1}^{N} v_i(\mathbf{x})\nabla^2 v_i(\mathbf{x})$$

Gauss-Newton Method

Approximate the Hessian matrix as:

$$\nabla^2 F(\mathbf{x}) \cong 2\mathbf{J}^T(\mathbf{x})\mathbf{J}(\mathbf{x})$$

Newton's method becomes:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - [2\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k)]^{-1} 2\mathbf{J}^T(\mathbf{x}_k)\mathbf{v}(\mathbf{x}_k)$$

$$= \mathbf{x}_k - [\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k)]^{-1} \mathbf{J}^T(\mathbf{x}_k)\mathbf{v}(\mathbf{x}_k)$$

Levenberg-Marquardt

Gauss-Newton approximates the Hessian by:

$$\mathbf{H} = \mathbf{J}^T\mathbf{J}$$

This matrix may be singular, but can be made invertible as follows:

$$\mathbf{G} = \mathbf{H} + \mu\mathbf{I}$$

If the eigenvalues and eigenvectors of **H** are:

$$\{\lambda_1, \lambda_2, \dots, \lambda_n\} \qquad \{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_n\}$$

then

$$\mathbf{G}\mathbf{z}_i = [\mathbf{H} + \mu\mathbf{I}]\mathbf{z}_i = \mathbf{H}\mathbf{z}_i + \mu\mathbf{z}_i = \lambda_i\mathbf{z}_i + \mu\mathbf{z}_i = \overbrace{(\lambda_i + \mu)}^{\text{Eigenvalues of } \mathbf{G}}\mathbf{z}_i$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k - [\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k) + \mu_k\mathbf{I}]^{-1}\mathbf{J}^T(\mathbf{x}_k)\mathbf{v}(\mathbf{x}_k)$$

Adjustment of μ_k

As $\mu_k \rightarrow 0$, LM becomes Gauss-Newton.

$$\mathbf{x}_{k+1} = \mathbf{x}_k - [\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k)]^{-1}\mathbf{J}^T(\mathbf{x}_k)\mathbf{v}(\mathbf{x}_k)$$

As $\mu_k \rightarrow \infty$, LM becomes Steepest Descent with small learning rate.

$$\mathbf{x}_{k+1} \cong \mathbf{x}_k - \frac{1}{\mu_k}\mathbf{J}^T(\mathbf{x}_k)\mathbf{v}(\mathbf{x}_k) = \mathbf{x}_k - \frac{1}{2\mu_k}\nabla \mathrm{F}(\mathbf{x})$$

Therefore, begin with a small μ_k to use Gauss-Newton and speed convergence. If a step does not yield a smaller $F(\mathbf{x})$, then repeat the step with an increased μ_k until $F(\mathbf{x})$ is decreased. $F(\mathbf{x})$ must decrease eventually, since we will be taking a very small step in the steepest descent direction.

Application to Multilayer Network

The performance index for the multilayer network is:

$$F(\mathbf{x}) = \sum_{q=1}^{Q} (\mathbf{t}_q - \mathbf{a}_q)^T (\mathbf{t}_q - \mathbf{a}_q) = \sum_{q=1}^{Q} \mathbf{e}_q^T \mathbf{e}_q = \sum_{q=1}^{Q} \sum_{j=1}^{S^M} (e_{j,q})^2 = \sum_{i=1}^{N} (v_i)^2$$

The error vector is:

$$\mathbf{v}^T = \begin{bmatrix} v_1 & v_2 & \dots & v_N \end{bmatrix} = \begin{bmatrix} e_{1,1} & e_{2,1} & \dots & e_{S^M,1} & e_{1,2} & \dots & e_{S^M,Q} \end{bmatrix}$$

The parameter vector is:

$$\mathbf{x}^T = \begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix} = \begin{bmatrix} w^1_{1,1} & w^1_{1,2} & \dots & w^1_{S^1,R} & b^1_1 & \dots & b^1_{S^1} & \mathrm{w}^2_{1,1} & \dots & b^M_{S^M} \end{bmatrix}$$

The dimensions of the two vectors are:

$$N = Q \times S^M \qquad n = S^1(R+1) + S^2(S^1+1) + \dots + S^M(S^{M-1}+1)$$

Jacobian Matrix

$$\mathbf{J}(\mathbf{x}) = \begin{bmatrix} \frac{\partial e_{1,1}}{\partial w^1_{1,1}} & \frac{\partial e_{1,1}}{\partial w^1_{1,2}} & \cdots & \frac{\partial e_{1,1}}{\partial w^1_{S^1,R}} & \frac{\partial e_{1,1}}{\partial b^1_1} & \cdots \\ \frac{\partial e_{2,1}}{\partial w^1_{1,1}} & \frac{\partial e_{2,1}}{\partial w^1_{1,2}} & \cdots & \frac{\partial e_{2,1}}{\partial w^1_{S^1,R}} & \frac{\partial e_{2,1}}{\partial b^1_1} & \cdots \\ \vdots & \vdots & & \vdots & \vdots & \\ \frac{\partial e_{S^M,1}}{\partial w^1_{1,1}} & \frac{\partial e_{S^M,1}}{\partial w^1_{1,2}} & \cdots & \frac{\partial \mathrm{e}_{e_{S^M,1}}}{\partial w^1_{S^1,R}} & \frac{\partial \mathrm{e}_{e_{S^M,1}}}{\partial b^1_1} & \cdots \\ \frac{\partial e_{1,2}}{\partial w^1_{1,1}} & \frac{\partial e_{1,2}}{\partial w^1_{1,2}} & \cdots & \frac{\partial e_{1,2}}{\partial w^1_{S^1,R}} & \frac{\partial e_{1,2}}{\partial b^1_1} & \cdots \\ \vdots & \vdots & & \vdots & \vdots & \end{bmatrix}$$

Computing the Jacobian

SDBP computes terms like:

$$\frac{\partial \hat{F}(\mathbf{x})}{\partial x_l} = \frac{\partial \mathbf{e}_q^T \mathbf{e}_q}{\partial x_l}$$

using the chain rule:

$$\frac{\partial \hat{F}}{\partial w_{i,j}^m} = \frac{\partial \hat{F}}{\partial n_i^m} \times \frac{\partial n_i^m}{\partial w_{i,j}^m}$$

where the sensitivity

$$s_i^m \equiv \frac{\partial \hat{F}}{\partial n_i^m}$$

is computed using backpropagation.

For the Jacobian we need to compute terms like:

$$[\mathbf{J}]_{h,l} = \frac{\partial v_h}{\partial x_l} = \frac{\partial e_{k,q}}{\partial x_l}$$

Marquardt Sensitivity

If we define a Marquardt sensitivity:

$$\tilde{s}^m_{i,h} \equiv \frac{\partial v_h}{\partial n^m_{i,q}} = \frac{\partial e_{k,q}}{\partial n^m_{i,q}} \qquad h = (q-1)S^M + k$$

We can compute the Jacobian as follows:

weight

$$[\mathbf{J}]_{h,l} = \frac{\partial v_h}{\partial x_l} = \frac{\partial e_{k,q}}{\partial w^m_{i,j}} = \frac{\partial e_{k,q}}{\partial n^m_{i,q}} \times \frac{\partial n^m_{i,q}}{\partial w^m_{i,j}} = \tilde{s}^m_{i,h} \times \frac{\partial n^m_{i,q}}{\partial w^m_{i,j}} = \tilde{s}^m_{i,h} \times a^{m-1}_{j,q}$$

bias

$$[\mathbf{J}]_{h,l} = \frac{\partial v_h}{\partial x_l} = \frac{\partial e_{k,q}}{\partial b^m_i} = \frac{\partial e_{k,q}}{\partial n^m_{i,q}} \times \frac{\partial n^m_{i,q}}{\partial b^m_i} = \tilde{s}^m_{i,h} \times \frac{\partial n^m_{i,q}}{\partial b^m_i} = \tilde{s}^m_{i,h}$$

Computing the Sensitivities

Initialization

$$\tilde{s}^M_{i,h} = \frac{\partial v_h}{\partial n^M_{i,q}} = \frac{\partial e_{k,q}}{\partial n^M_{i,q}} = \frac{\partial(\mathrm{t}_{k,q} - a^M_{k,q})}{\partial n^M_{i,q}} = -\frac{\partial \mathrm{a}^M_{k,q}}{\partial n^M_{i,q}}$$

$$\tilde{s}^M_{i,h} = \begin{cases} -\dot{f}^M(n^M_{i,q}) & \text{for } i = k \\ 0 & \text{for } i \neq k \end{cases}$$

$$\tilde{\mathbf{S}}^M_q = -\dot{\mathbf{F}}^M(\mathbf{n}^M_q)$$

Backpropagation

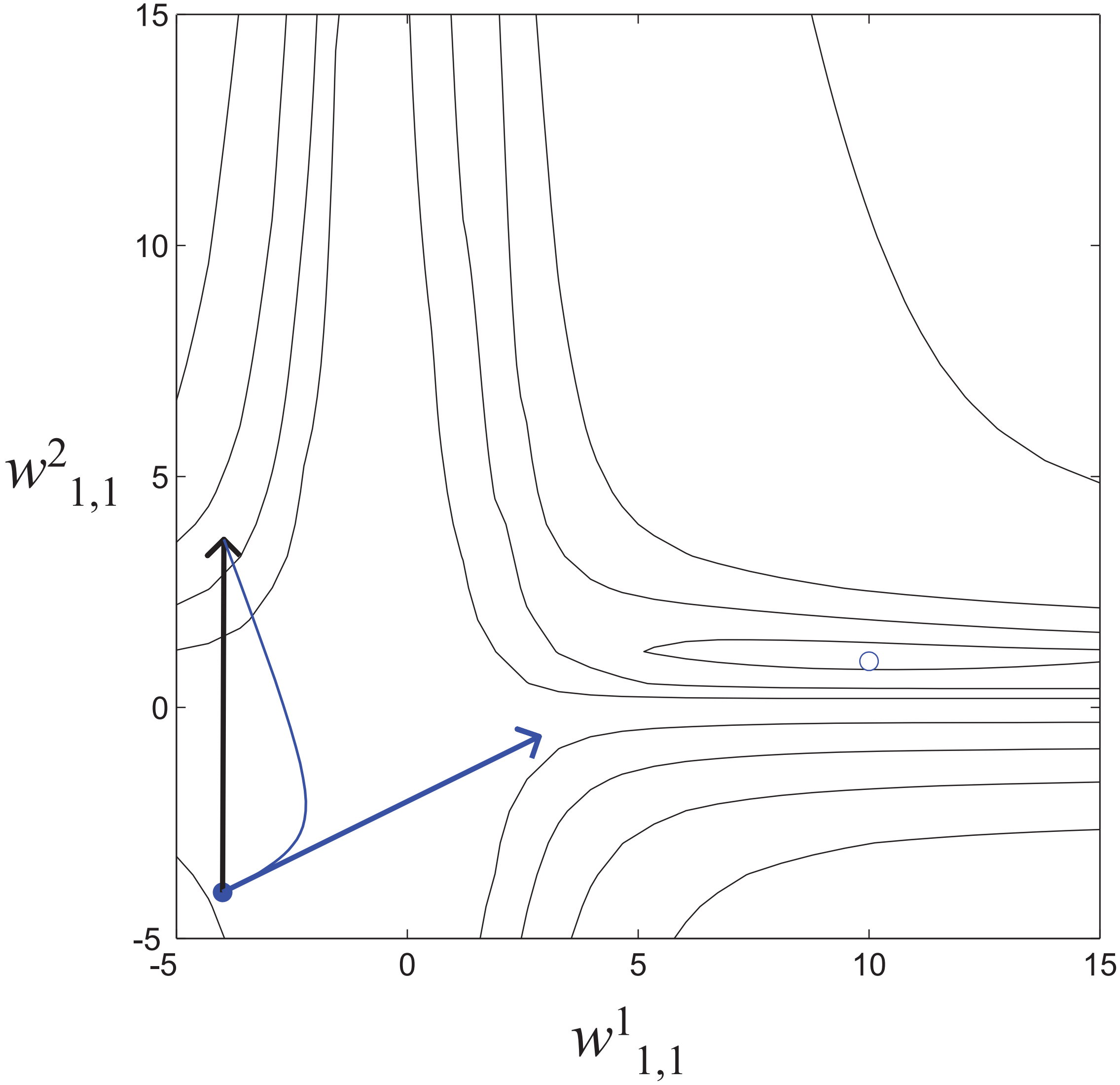
$$\tilde{\mathbf{S}}^m_q = \dot{\mathbf{F}}^m(\mathbf{n}^m_q)(\mathbf{W}^{m+1})^T\tilde{\mathbf{S}}^{m+1}_q$$

$$\tilde{\mathbf{S}}^m = \left[\tilde{\mathbf{S}}^m_1 \middle| \tilde{\mathbf{S}}^m_2 \middle| \ldots \middle| \tilde{\mathbf{S}}^m_{\mathrm{Q}}\right]$$

LMBP

- Present all inputs to the network and compute the corresponding network outputs and the errors. Compute the sum of squared errors over all inputs.
- Compute the Jacobian matrix. Calculate the sensitivities with the backpropagation algorithm, after initializing. Augment the individual matrices into the Marquardt sensitivities. Compute the elements of the Jacobian matrix.
- Solve to obtain the change in the weights.
- Recompute the sum of squared errors with the new weights. If this new sum of squares is smaller than that computed in step 1, then divide μ_k by υ, update the weights and go back to step 1. If the sum of squares is not reduced, then multiply μ_k by υ and go back to step 3.

Example LMBP Step



LMBP Trajectory